

# PROGRAM EXECUTION METHOD

## BACKGROUND OF THE INVENTION:

### 1. Field of the Invention

5  
SUSA The present invention relates to a program execution method for dynamically compiling a program that is created using a programming language, such as Java.

### 10 2. Description of the Related Art

15  
20  
25  
30  
35  
40  
45  
50  
55  
60  
65  
70  
75  
80  
85  
90  
95  
100  
105  
110  
115  
120  
125  
130  
135  
140  
145  
150  
155  
160  
165  
170  
175  
180  
185  
190  
195  
200  
205  
210  
215  
220  
225  
230  
235  
240  
245  
250  
255  
260  
265  
270  
275  
280  
285  
290  
295  
300  
305  
310  
315  
320  
325  
330  
335  
340  
345  
350  
355  
360  
365  
370  
375  
380  
385  
390  
395  
400  
405  
410  
415  
420  
425  
430  
435  
440  
445  
450  
455  
460  
465  
470  
475  
480  
485  
490  
495  
500  
505  
510  
515  
520  
525  
530  
535  
540  
545  
550  
555  
560  
565  
570  
575  
580  
585  
590  
595  
600  
605  
610  
615  
620  
625  
630  
635  
640  
645  
650  
655  
660  
665  
670  
675  
680  
685  
690  
695  
700  
705  
710  
715  
720  
725  
730  
735  
740  
745  
750  
755  
760  
765  
770  
775  
780  
785  
790  
795  
800  
805  
810  
815  
820  
825  
830  
835  
840  
845  
850  
855  
860  
865  
870  
875  
880  
885  
890  
895  
900  
905  
910  
915  
920  
925  
930  
935  
940  
945  
950  
955  
960  
965  
970  
975  
980  
985  
990  
995  
1000  
1005  
1010  
1015  
1020  
1025  
1030  
1035  
1040  
1045  
1050  
1055  
1060  
1065  
1070  
1075  
1080  
1085  
1090  
1095  
1100  
1105  
1110  
1115  
1120  
1125  
1130  
1135  
1140  
1145  
1150  
1155  
1160  
1165  
1170  
1175  
1180  
1185  
1190  
1195  
1200  
1205  
1210  
1215  
1220  
1225  
1230  
1235  
1240  
1245  
1250  
1255  
1260  
1265  
1270  
1275  
1280  
1285  
1290  
1295  
1300  
1305  
1310  
1315  
1320  
1325  
1330  
1335  
1340  
1345  
1350  
1355  
1360  
1365  
1370  
1375  
1380  
1385  
1390  
1395  
1400  
1405  
1410  
1415  
1420  
1425  
1430  
1435  
1440  
1445  
1450  
1455  
1460  
1465  
1470  
1475  
1480  
1485  
1490  
1495  
1500  
1505  
1510  
1515  
1520  
1525  
1530  
1535  
1540  
1545  
1550  
1555  
1560  
1565  
1570  
1575  
1580  
1585  
1590  
1595  
1600  
1605  
1610  
1615  
1620  
1625  
1630  
1635  
1640  
1645  
1650  
1655  
1660  
1665  
1670  
1675  
1680  
1685  
1690  
1695  
1700  
1705  
1710  
1715  
1720  
1725  
1730  
1735  
1740  
1745  
1750  
1755  
1760  
1765  
1770  
1775  
1780  
1785  
1790  
1795  
1800  
1805  
1810  
1815  
1820  
1825  
1830  
1835  
1840  
1845  
1850  
1855  
1860  
1865  
1870  
1875  
1880  
1885  
1890  
1895  
1900  
1905  
1910  
1915  
1920  
1925  
1930  
1935  
1940  
1945  
1950  
1955  
1960  
1965  
1970  
1975  
1980  
1985  
1990  
1995  
2000  
2005  
2010  
2015  
2020  
2025  
2030  
2035  
2040  
2045  
2050  
2055  
2060  
2065  
2070  
2075  
2080  
2085  
2090  
2095  
2100  
2105  
2110  
2115  
2120  
2125  
2130  
2135  
2140  
2145  
2150  
2155  
2160  
2165  
2170  
2175  
2180  
2185  
2190  
2195  
2200  
2205  
2210  
2215  
2220  
2225  
2230  
2235  
2240  
2245  
2250  
2255  
2260  
2265  
2270  
2275  
2280  
2285  
2290  
2295  
2300  
2305  
2310  
2315  
2320  
2325  
2330  
2335  
2340  
2345  
2350  
2355  
2360  
2365  
2370  
2375  
2380  
2385  
2390  
2395  
2400  
2405  
2410  
2415  
2420  
2425  
2430  
2435  
2440  
2445  
2450  
2455  
2460  
2465  
2470  
2475  
2480  
2485  
2490  
2495  
2500  
2505  
2510  
2515  
2520  
2525  
2530  
2535  
2540  
2545  
2550  
2555  
2560  
2565  
2570  
2575  
2580  
2585  
2590  
2595  
2600  
2605  
2610  
2615  
2620  
2625  
2630  
2635  
2640  
2645  
2650  
2655  
2660  
2665  
2670  
2675  
2680  
2685  
2690  
2695  
2700  
2705  
2710  
2715  
2720  
2725  
2730  
2735  
2740  
2745  
2750  
2755  
2760  
2765  
2770  
2775  
2780  
2785  
2790  
2795  
2800  
2805  
2810  
2815  
2820  
2825  
2830  
2835  
2840  
2845  
2850  
2855  
2860  
2865  
2870  
2875  
2880  
2885  
2890  
2895  
2900  
2905  
2910  
2915  
2920  
2925  
2930  
2935  
2940  
2945  
2950  
2955  
2960  
2965  
2970  
2975  
2980  
2985  
2990  
2995  
3000  
3005  
3010  
3015  
3020  
3025  
3030  
3035  
3040  
3045  
3050  
3055  
3060  
3065  
3070  
3075  
3080  
3085  
3090  
3095  
3100  
3105  
3110  
3115  
3120  
3125  
3130  
3135  
3140  
3145  
3150  
3155  
3160  
3165  
3170  
3175  
3180  
3185  
3190  
3195  
3200  
3205  
3210  
3215  
3220  
3225  
3230  
3235  
3240  
3245  
3250  
3255  
3260  
3265  
3270  
3275  
3280  
3285  
3290  
3295  
3300  
3305  
3310  
3315  
3320  
3325  
3330  
3335  
3340  
3345  
3350  
3355  
3360  
3365  
3370  
3375  
3380  
3385  
3390  
3395  
3400  
3405  
3410  
3415  
3420  
3425  
3430  
3435  
3440  
3445  
3450  
3455  
3460  
3465  
3470  
3475  
3480  
3485  
3490  
3495  
3500  
3505  
3510  
3515  
3520  
3525  
3530  
3535  
3540  
3545  
3550  
3555  
3560  
3565  
3570  
3575  
3580  
3585  
3590  
3595  
3600  
3605  
3610  
3615  
3620  
3625  
3630  
3635  
3640  
3645  
3650  
3655  
3660  
3665  
3670  
3675  
3680  
3685  
3690  
3695  
3700  
3705  
3710  
3715  
3720  
3725  
3730  
3735  
3740  
3745  
3750  
3755  
3760  
3765  
3770  
3775  
3780  
3785  
3790  
3795  
3800  
3805  
3810  
3815  
3820  
3825  
3830  
3835  
3840  
3845  
3850  
3855  
3860  
3865  
3870  
3875  
3880  
3885  
3890  
3895  
3900  
3905  
3910  
3915  
3920  
3925  
3930  
3935  
3940  
3945  
3950  
3955  
3960  
3965  
3970  
3975  
3980  
3985  
3990  
3995  
4000  
4005  
4010  
4015  
4020  
4025  
4030  
4035  
4040  
4045  
4050  
4055  
4060  
4065  
4070  
4075  
4080  
4085  
4090  
4095  
4100  
4105  
4110  
4115  
4120  
4125  
4130  
4135  
4140  
4145  
4150  
4155  
4160  
4165  
4170  
4175  
4180  
4185  
4190  
4195  
4200  
4205  
4210  
4215  
4220  
4225  
4230  
4235  
4240  
4245  
4250  
4255  
4260  
4265  
4270  
4275  
4280  
4285  
4290  
4295  
4300  
4305  
4310  
4315  
4320  
4325  
4330  
4335  
4340  
4345  
4350  
4355  
4360  
4365  
4370  
4375  
4380  
4385  
4390  
4395  
4400  
4405  
4410  
4415  
4420  
4425  
4430  
4435  
4440  
4445  
4450  
4455  
4460  
4465  
4470  
4475  
4480  
4485  
4490  
4495  
4500  
4505  
4510  
4515  
4520  
4525  
4530  
4535  
4540  
4545  
4550  
4555  
4560  
4565  
4570  
4575  
4580  
4585  
4590  
4595  
4600  
4605  
4610  
4615  
4620  
4625  
4630  
4635  
4640  
4645  
4650  
4655  
4660  
4665  
4670  
4675  
4680  
4685  
4690  
4695  
4700  
4705  
4710  
4715  
4720  
4725  
4730  
4735  
4740  
4745  
4750  
4755  
4760  
4765  
4770  
4775  
4780  
4785  
4790  
4795  
4800  
4805  
4810  
4815  
4820  
4825  
4830  
4835  
4840  
4845  
4850  
4855  
4860  
4865  
4870  
4875  
4880  
4885  
4890  
4895  
4900  
4905  
4910  
4915  
4920  
4925  
4930  
4935  
4940  
4945  
4950  
4955  
4960  
4965  
4970  
4975  
4980  
4985  
4990  
4995  
5000  
5005  
5010  
5015  
5020  
5025  
5030  
5035  
5040  
5045  
5050  
5055  
5060  
5065  
5070  
5075  
5080  
5085  
5090  
5095  
5100  
5105  
5110  
5115  
5120  
5125  
5130  
5135  
5140  
5145  
5150  
5155  
5160  
5165  
5170  
5175  
5180  
5185  
5190  
5195  
5200  
5205  
5210  
5215  
5220  
5225  
5230  
5235  
5240  
5245  
5250  
5255  
5260  
5265  
5270  
5275  
5280  
5285  
5290  
5295  
5300  
5305  
5310  
5315  
5320  
5325  
5330  
5335  
5340  
5345  
5350  
5355  
5360  
5365  
5370  
5375  
5380  
5385  
5390  
5395  
5400  
5405  
5410  
5415  
5420  
5425  
5430  
5435  
5440  
5445  
5450  
5455  
5460  
5465  
5470  
5475  
5480  
5485  
5490  
5495  
5500  
5505  
5510  
5515  
5520  
5525  
5530  
5535  
5540  
5545  
5550  
5555  
5560  
5565  
5570  
5575  
5580  
5585  
5590  
5595  
5600  
5605  
5610  
5615  
5620  
5625  
5630  
5635  
5640  
5645  
5650  
5655  
5660  
5665  
5670  
5675  
5680  
5685  
5690  
5695  
5700  
5705  
5710  
5715  
5720  
5725  
5730  
5735  
5740  
5745  
5750  
5755  
5760  
5765  
5770  
5775  
5780  
5785  
5790  
5795  
5800  
5805  
5810  
5815  
5820  
5825  
5830  
5835  
5840  
5845  
5850  
5855  
5860  
5865  
5870  
5875  
5880  
5885  
5890  
5895  
5900  
5905  
5910  
5915  
5920  
5925  
5930  
5935  
5940  
5945  
5950  
5955  
5960  
5965  
5970  
5975  
5980  
5985  
5990  
5995  
6000  
6005  
6010  
6015  
6020  
6025  
6030  
6035  
6040  
6045  
6050  
6055  
6060  
6065  
6070  
6075  
6080  
6085  
6090  
6095  
6100  
6105  
6110  
6115  
6120  
6125  
6130  
6135  
6140  
6145  
6150  
6155  
6160  
6165  
6170  
6175  
6180  
6185  
6190  
6195  
6200  
6205  
6210  
6215  
6220  
6225  
6230  
6235  
6240  
6245  
6250  
6255  
6260  
6265  
6270  
6275  
6280  
6285  
6290  
6295  
6300  
6305  
6310  
6315  
6320  
6325  
6330  
6335  
6340  
6345  
6350  
6355  
6360  
6365  
6370  
6375  
6380  
6385  
6390  
6395  
6400  
6405  
6410  
6415  
6420  
6425  
6430  
6435  
6440  
6445  
6450  
6455  
6460  
6465  
6470  
6475  
6480  
6485  
6490  
6495  
6500  
6505  
6510  
6515  
6520  
6525  
6530  
6535  
6540  
6545  
6550  
6555  
6560  
6565  
6570  
6575  
6580  
6585  
6590  
6595  
6600  
6605  
6610  
6615  
6620  
6625  
6630  
6635  
6640  
6645  
6650  
6655  
6660  
6665  
6670  
6675  
6680  
6685  
6690  
6695  
6700  
6705  
6710  
6715  
6720  
6725  
6730  
6735  
6740  
6745  
6750  
6755  
6760  
6765  
6770  
6775  
6780  
6785  
6790  
6795  
6800  
6805  
6810  
6815  
6820  
6825  
6830  
6835  
6840  
6845  
6850  
6855  
6860  
6865  
6870  
6875  
6880  
6885  
6890  
6895  
6900  
6905  
6910  
6915  
6920  
6925  
6930  
6935  
6940  
6945  
6950  
6955  
6960  
6965  
6970  
6975  
6980  
6985  
6990  
6995  
7000  
7005  
7010  
7015  
7020  
7025  
7030  
7035  
7040  
7045  
7050  
7055  
7060  
7065  
7070  
7075  
7080  
7085  
7090  
7095  
7100  
7105  
7110  
7115  
7120  
7125  
7130  
7135  
7140  
7145  
7150  
7155  
7160  
7165  
7170  
7175  
7180  
7185  
7190  
7195  
7200  
7205  
7210  
7215  
7220  
7225  
7230  
7235  
7240  
7245  
7250  
7255  
7260  
7265  
7270  
7275  
7280  
7285  
7290  
7295  
7300  
7305  
7310  
7315  
7320  
7325  
7330  
7335  
7340  
7345  
7350  
7355  
7360  
7365  
7370  
7375  
7380  
7385  
7390  
7395  
7400  
7405  
7410  
7415  
7420  
7425  
7430  
7435  
7440  
7445  
7450  
7455  
7460  
7465  
7470  
7475  
7480  
7485  
7490  
7495  
7500  
7505  
7510  
7515  
7520  
7525  
7530  
7535  
7540  
7545  
7550  
7555  
7560  
7565  
7570  
7575  
7580  
7585  
7590  
7595  
7600  
7605  
7610  
7615  
7620  
7625  
7630  
7635  
7640  
7645  
7650  
7655  
7660  
7665  
7670  
7675  
7680  
7685  
7690  
7695  
7700  
7705  
7710  
7715  
7720  
7725  
7730  
7735  
7740  
7745  
7750  
7755  
7760  
7765  
7770  
7775  
7780  
7785  
7790  
7795  
7800  
7805  
7810  
7815  
7820  
7825  
7830  
7835  
7840  
7845  
7850  
7855  
7860  
7865  
7870  
7875  
7880  
7885  
7890  
7895  
7900  
7905  
7910  
7915  
7920  
7925  
7930  
7935  
7940  
7945  
7950  
7955  
7960  
7965  
7970  
7975  
7980  
7985  
7990  
7995  
8000  
8005  
8010  
8015  
8020  
8025  
8030  
8035  
8040  
8045  
8050  
8055  
8060  
8065  
8070  
8075  
8080  
8085  
8090  
8095  
8100  
8105  
8110  
8115  
8120  
8125  
8130  
8135  
8140  
8145  
8150  
8155  
8160  
8165  
8170  
8175  
8180  
8185  
8190  
8195  
8200  
8205  
8210  
8215  
8220  
8225  
8230  
8235  
8240  
8245  
8250  
8255  
8260  
8265  
8270  
8275  
8280  
8285  
8290  
8295  
8300  
8305  
8310  
8315  
8320  
8325  
8330  
8335  
8340  
8345  
8350  
8355  
8360  
8365  
8370  
8375  
8380  
8385  
8390  
8395  
8400  
8405  
8410  
8415  
8420  
8425  
8430  
8435  
8440  
8445  
8450  
8455  
8460  
8465  
8470  
8475  
8480  
8485  
8490  
8495  
8500  
8505  
8510  
8515  
8520  
8525  
8530  
8535  
8540  
8545  
8550  
8555  
8560  
8565  
8570  
8575  
8580  
8585  
8590  
8595  
8600  
8605  
8610  
8615  
8620  
8625  
8630  
8635  
8640  
8645  
8650  
8655  
8660  
8665  
8670  
8675  
8680  
8685  
8690  
8695  
8700  
8705  
8710  
8715  
8720  
8725  
8730  
8735  
8740  
8745  
8750  
8755  
8760  
8765  
8770  
8775  
8780  
8785  
8790  
8795  
8800  
8805  
8810  
8815  
8820  
8825  
8830  
8835  
8840  
8845  
8850  
8855  
8860  
8865  
8870  
8875  
8880  
8885  
8890  
8895  
8900  
8905  
8910  
8915  
8920  
8925  
8930  
8935  
8940  
8945  
8950  
8955  
8960  
8965  
8970  
8975  
8980  
8985  
8990  
8995  
9000  
9005  
9010  
9015  
9020  
9025  
9030  
9035  
9040  
9045  
9050  
9055  
9060  
9065  
9070  
9075  
9080  
9085  
9090  
9095  
9100  
9105  
9110  
9115  
9120  
9125  
9130  
9135  
9140  
9145  
9150  
9155  
9160  
9165  
9170  
9175  
9180  
9185  
9190  
9195  
9200  
9205  
9210  
9215  
9220  
9225  
9230  
9235  
9240  
9245  
9250  
9255  
9260  
9265  
9270  
9275  
9280  
9285  
9290  
9295  
9300  
9305  
9310  
9315  
9320  
9325  
9330  
9335  
9340  
9345  
9350  
9355  
9360  
9365  
9370  
9375  
9380  
9385  
9390  
9395  
9400  
9405  
9410  
9415  
9420  
9425  
9430  
9435  
9440  
9445  
9450  
9455  
9460  
9465  
9470  
9475  
9480  
9485  
9490  
9495  
9500  
9505  
9510  
9515  
9520  
9525  
9530  
9535  
9540  
9545  
9550  
9555  
9560  
9565  
9570  
9575  
9580  
9585  
9590  
9595  
9600  
9605  
9610  
9615  
9620  
9625  
9630  
9635  
9640  
9645  
9650  
9655  
9660  
9665  
9670  
9675  
9680  
9685  
9690  
9695  
9700  
9705  
9710  
9715  
9720  
9725  
9730  
9735  
9740  
9745  
9750  
9755  
9760  
9765  
9770  
9775  
9780  
9785  
9790  
9795  
9800  
9805  
9810  
9815  
9820  
9825  
9830  
9835  
9840  
9845  
9850  
9855  
9860  
9865  
9870  
9875  
9880  
9885  
9890  
9895  
9900  
9905  
9910  
9915  
9920  
9925  
9930  
9935  
9940  
9945  
9950  
9955  
9960  
9965  
9970  
9975  
9980  
9985  
9990  
9995  
10000  
10005  
10010  
10015  
10020  
10025  
10030  
10035  
10040  
10045  
10050  
10055  
10060  
10065  
10070  
10075  
10080  
10085  
10090  
10095  
10100  
10105  
10110  
10115  
10120  
10125  
10130  
10135  
10140  
10145  
10150  
10155  
10160  
10165  
10170  
10175  
10180  
10185  
10190  
10195  
10200  
10205  
10210  
10215  
10220  
10225  
10230  
10235  
10240  
10245  
1

According to this method, compiling is performed substantially without optimization before a program is run the first time, and an invocation counter is employed to count the times the program is activated. Thereafter, when the activation count for a program exceeds a threshold value, an optimization count is incremented and recompiling is performed.

5

Since with this method a recompiled program is executed only at the time at which the pertinent method is called, the conventional method is not effective for a program that, even though it is activated only once, includes a loop or loops for which iterations are repeatedly performed, and that for its execution requires an extended period of time (includes a loop to be optimized, and a loop for which multiple iterations are performed).

10

A conventional method called deoptimization is also well known, as described in "Debugging Optimized Code With Dynamic Deoptimization," Urs Hoelzle, Craig Chambers and David Ungar, in Proceedings of the SIGPLAN '92 Conference on Programming Language Design and Implementation, pp. 21-38, June 1992 [HCU92].

15

According to this method, unoptimized code that is created to debug code for an optimized method, and to perform the debugging of the execution program, is transferred to the obtained code. With this method, (1) before compiling is performed for optimization, interrupt points are designated at two positions: at the beginning of the method and at a branch command near the end of a loop; (2) optimization compiling is performed with an additional restriction that optimization that exceeds an interrupt point will not be performed; (3) when an interrupt is issued during program execution, the execution of the program is resumed following each interrupt point; and (4) a stack frame for the interrupt points is stored. Then, (5) a method is compiled without being optimized; and (6) the stack frame is reproduced in consonance with the non-optimized state, and the process enters an interrupt state. This method is similar to the present invention for transferring the process during the execution of a program. However, since a transfer target is un-optimized code, and since the execution condition, such as a register image, at the transfer point can be uniquely determined using source code, this

20

25

conventional method can not be applied directly for a transfer to optimized code for which the process at the transfer destination becomes complex.

Further, a conventional well known method is called dynamic recompilation, as described in  
5 "Optimizing Dynamical-Dispatched Calls With Run-Time Type Feedback," Urs Hoelzle and David Ungar, in Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation, pp. 326-336, published as SIGPLAN Notices 29(6), June 1994 [HU94].

10 This method is used to provide an explanation and an overview of a method whereby, during the execution of a non-optimized method, the pertinent method is replaced by one that was optimized when it was compiled. According to this method, (1) a method to be recompiled is detected, (2) a mark is set at a point at which execution is to be resumed, and (3) values that  
15 are pointed to by all the currently effective registers are calculated. When these processes have been successfully performed, (4) a non-optimized method in a stack is replaced by an optimized one. If such a replacement is disabled, generated code is called by a succeeding method and is employed. While detailed processing has not been explained for the above method, the following problems have arisen because, as it is described, the processing is the reverse of that performed in deoptimization [HCU92].

20 The problems with the above method are that a plurality of transfer points are not handled; that optimization passing beyond a transfer point is not performed; and that an increase in memory is not taken into account.

25 Two methods are available to switch the execution of a program when optimization compiling is performed: a method whereby an execution process is changed by calling the next method, and a method whereby a method that is being executed is replaced by optimized code. The second method, especially, is a function required to increase the processing speed for a method that includes a loop for which multiple iterations are performed, even though it is activated only  
30 once, and that for its execution requires an extended period of time. However, when the

conventional method, whereby the methods are switched during execution, is employed, the following two problems are encountered.

First, optimization does not pass beyond a position (hereinafter referred to a transfer point) in a method that is to be switched. Therefore, if a transfer point is present, the quality of code obtained is lower than the code that is generated when there is no transfer point. The execution speed for a program when a succeeding method is called is lower than the execution speed obtained when optimization compiling is performed while no transfer point is present.

Especially when a transfer point has been inserted into a loop, optimization of the loop, which can greatly affect the execution performance, can not be performed, and as a result, considerable degradation of the performance occurs.

Second, no consideration is given to a case in a multithreaded environment in which processes are replaced at a plurality of different transfer points, and compiling is performed each time a transfer occurs. These multiple compiling sessions contribute to the deterioration of the execution speed; generated codes overlap and the consumption of memory is increased.

## SUMMARY OF THE INVENTION

To resolve these shortcomings, it is one object of the present invention to provide a program execution method whereby higher optimization that pass beyond a transfer point can be performed, even when in a program a change is made at the transfer point.

It is another object of the present invention to provide a program execution method whereby, even when processes at multiple transfer points in a multithreaded environment are transferred, multiple processing sessions for compiling are not required, and generated codes do not overlap.

To achieve the above objects, according to the present invention, a program execution method, for transferring, from an interpreter process to a compiled code process, a method that is currently being executed for code that includes a plurality of transfer points, comprises the steps of: moving the transfer points for code to the top of a loop process when no problem occurs, even when the transfer points are moved to the top of the loop process; copying to a location immediately preceding the loop process, when the transfer points are located inside the loop process, a point that post-dominates the top of the loop process and the transfer points; storing information for generating recalculation code for specific transfer points when the moving of the code and privatization and a common sub-expression elimination that are performed pass beyond the specific transfer points; and performing a recalculation during a transfer process.

Preferably, the program execution method further comprises a step of: defining as a new transfer point, a point from the interpreter process to the compiled code process whereat, when the method that is currently being executed is replaced, the execution speed is increased compared with when the execution is not transferred.

Preferably, the program execution method further comprises the steps of: generating information required to perform a transfer from the interpreter process to the compiled code process; and storing the obtained information while correlating the obtained information with the transfer points, wherein, at the recalculation step, the information stored for the transfer points is employed.

According to the present invention, a program for transferring, from an interpreter process to a compiled code process, a method that is currently being executed for code that includes a plurality of transfer points, comprises the steps of: moving the transfer points for code to the top of a loop process when no problem occurs, even when the transfer points are moved to the top of the loop process; copying to a location immediately preceding the loop process, when the transfer points are located inside the loop process, a point that post-dominates the top of the loop process and the transfer points; storing information for generating recalculation code

for specific transfer points when the moving of the code and privatization and a common sub-expression elimination that are performed pass beyond the specific transfer points; and performing a recalculation during a transfer process.

5

## BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram showing the configuration of a program execution apparatus according to the present invention.

10

Fig. 2 is a first flowchart showing the processing performed by the program execution apparatus according to the present invention.

15

Fig. 3 is a second flowchart showing the processing performed by the program execution apparatus according to the present invention.

Fig. 4 is a third flowchart showing the processing performed by the program execution apparatus according to the present invention.

20

Fig. 5 is a fourth flowchart showing the processing performed by the program execution apparatus according to the present invention.

Fig. 6 is a fifth flowchart showing the processing performed by the program execution apparatus according to the present invention.

25

Fig. 7 is a sixth flowchart showing the processing performed by the program execution apparatus according to the present invention.

Fig. 8 is a seventh flowchart showing the processing performed by the program execution apparatus according to the present invention.

30

## DESCRIPTION OF THE PREFERRED EMBODIMENT

The essential points of the present invention will now be described.

### 5 Method for preventing deterioration of code quality due to the presence of a transfer point

10 When a transfer point is present, there are two reasons the quality of code is deteriorated. One is that many loop optimization processes can not be performed when transfer points are present in a loop. The other is that an optimization method, such as the moving of code, and privatization or common sub-expression elimination, for storing a state in the middle of a calculation in a register or a local area in a memory, can not be performed. The following methods are employed to resolve these problems.

- 15 1. If no execution problem occurs, even when a transfer point is moved to the top of a loop, the transfer point in compiled code is moved to the top of a loop.
- 20 2. When the transfer point is located inside the loop, the process from the top of the loop to a point that post-dominates the top of the loop and the transfer point is copied to a location immediately preceding the top of the loop, and the loop is so modified that the transfer point is located outside the loop or at the top of the loop. As a result, the presence of the transfer point does not adversely affect the optimization of the loop.
- 25 3. When the movement of code and the privatization of the common sub-expression elimination are performed above the transfer point, information for generating recalculated code for the pertinent transfer point is stored at the transfer point. During a transfer process, recalculation is performed.

### 30 Method for employing code to cope with a normal call and a transfer in the middle of the execution of a program

A point that is to be transferred from the interpreter, and a point that will be transferred as a result of examining a method of a program and that will, as a result of the transfer, increase the processing speed are detected, and are defined as transfer points.

5

For each transfer point, transfer information, consisting of the register use state at a current transfer, point and recalculation code information are generated. In consonance with the addresses of the original code, a table including the transfer information is generated and is positioned at a location that the method can access. Upon the execution of the transfer process, the table is accessed by using the address of a transfer point as a key, and the transfer information is obtained.

10

There is one method whereby an unneeded variable area is not generated when a stack frame used by an interpreter is changed to a stack frame that is used by compiled code. According to this method, in the interpreter all the local variables that are designated by the individual methods are allocated to stack frames. However, in the compiled code, a restoration location is shared with a plurality of local variables, or a variable is provided that is stored only in a register and its value is not restored in the stack frame. Therefore, not all the local variables designated by the methods are allocated to the stack frames. To resolve this problem, the following procedure is employed.

15

20

A local variable area, in a stack frame that compiled code requires, is stored at a location that a method can access during a compiled code process. The size of this allocated area is no larger than the size required by compiled code. The transfer information includes information concerning whether a variable at a transfer point is stored in the register or in the stack frame. During the transfer process, first, a stack frame that the interpreter is using is restored, and a stack frame for compiled code is generated. Then, the transfer information is employed to copy the values of required local variables to registers or to the stack frame. Finally, the stack frame of the interpreter is removed, and the execution process is shifted to the transfer point.

25

30



When the interpreter process is changed only once to the execution of machine code, whereby the dynamic compiler is activated, buffer code is generated in a stack area to prevent an increase in the memory consumed by the buffer code.

5 The processing for the present invention will now be described.

Case wherein during the execution process an interpreter determines a transfer is required

10 When an interpreter determines during the execution process that a transfer is required, the following processing is performed.

- 11 1. If a method has already been compiled, the method is used to prepare a transfer  
12 information table to acquire transfer information from the address for a command at a  
13 transfer point. When the transfer information can be obtained, process 2 under  
14 "Interpreter process" below and the following processes are performed. When the  
15 transfer information can not be obtained, the transfer process is halted and the  
16 interpreter continues the execution processing.
- 17 2. A dynamic compiler is activated. At this time, the address of a transfer determination  
18 command and the address of a command to be executed after the transfer has been  
19 performed are transmitted to the compiler. When the address of a command to be  
20 executed after the transfer has been performed is obvious, this process may be  
21 eliminated.

25

Processing performed by a dynamic compiler

The following process is performed by a dynamic compiler.

1. The code for the method is divided to obtain a Basic Block (hereinafter simply referred to as a BB) and a control flow graph is generated. A transfer point designated by the interpreter is defined as the first in the BB.

2. A mark that includes the transfer point designated by the interpreter is provided for the BB.

3. The code for the method is examined to obtain a proposed transfer point, other than that designated by the interpreter, that can be transferred and that will provide a noticeable effect when the transfer is performed. This process is not performed when the defined transfer point is the point that was designated by the interpreter. The proposed transfer point is obtained by removing, from all the positions in the method that can be detected by the interpreter, locations at which the compiler can determine that no transfer effects will be provided. A point at which no transfer effect will be provided is, for example, a point outside a loop.

4. If no execution problem is encountered, even when the transfer point is moved to the top of the loop, the transfer point in the compiled code is moved to the top of the loop. A case where a problem in a transfer is encountered is one where a write command to a heap area is present, or one where a specific variable is defined between the top of a loop and the transfer point, and a reference for the variable is located between the two and the value of the variable can not be recalculated at the top of the loop. When recalculation can be performed, the recalculation information for the transfer point are generated.

5. If the transfer point is located inside the loop, the loop is modified so that the transfer point serves as the starting point for the loop (a loop to be optimized, including a multiplexed loop). As a result, the presence of the transfer point has no adverse affect on optimization. For this modification, code is copied from the entry point of the loop to the point in the loop that post-dominates the entry point and the transfer point. In a

worst case, the copied code is smaller than the code for the loop. Further, when the point to be transferred by the interpreter is limited to a branch command toward the end of a loop, only the control code for the loop need be copied in a Java "for" loop, while no code need be copied for a "do-while" loop.

6. When a process for moving code, or caching values using a common sub-expression or privatization, is performed that passes beyond a transfer point, the data for generating the recalculation code is stored for all the transfer points that are passed.
7. Code consonant with the method is generated.
8. The local variable area used by the compiled code is stored at a location that the method can access.
9. For each transfer point, transfer information are generated that include the code address of the transfer point, recalculation code data for caching, and register employment information for a transfer point. A transfer information table is prepared that is paired with the address of a transfer command in the original code. The table is then stored at a location that the method can access. The transfer information may be represented as actual code, or may be represented as data to be transmitted to the common transfer method. If the interpreter is changed to the execution of machine code only one time, whereat the dynamic compiler is activated, the transfer information can be generated in the stack area, so that the data can be immediately eliminated when the transfer has been completed.

#### Interpreter process

The following processes are performed by the interpreter:

1. When the dynamic compiling was successful, the transfer information is obtained from the transfer data table. When the compiling failed, the current transfer process is halted and program execution using the interpreter is continued.

2. The local variable area used for compiled code is obtained, necessary information in the stack frame that was employed by the interpreter is restored, and a stack frame for the compiled code is generated.

3. Necessary information is entered in the stack frame for the code that has been compiled using the transfer information. When the transfer information is represented as code, the code is executed. When the transfer information is represented as data, the data are transmitted to the transfer process method to begin the transfer processing. The transfer processing can be implemented by (1) copying, from a restoring area, a value to be stored in the stack area; (2) recalculating the value using the recalculation information, and entering the result in the local variable area, the register or the restoring area; and (3) loading the value into a register.

4. Program execution is begun at the address of the transfer point.

Program execution apparatus 1

Fig. 1 is a diagram illustrating the arrangement of a program execution apparatus 1 according to the present invention.

As is shown in Fig. 1, in the program execution apparatus 1 a server 10 and a client 12 are connected via a network 106.

The server 10 comprises Java source code 100, a Java bytecode compiler (JAVAC) 102 and Java bytecode 104.

The client 12 includes Java programs 120 and 130, which are received from a storage medium 140 and which are executed via a storage device 14 by hardware 132.

The Java program 120 includes a Java bytecode verifier 122, a Java interpreter 124, a JIT compiler 126 and a native code execution block 128.

In the server 10, the Java source code 100 is converted into bytecode 104 by a bytecode compiler, such as the JAVAC 102, and the obtained bytecode is transmitted via the network 106 to the client 12.

In the client 12, the bytecode verifier 122 verifies the bytecode 104 received from the server 10.

The Java interpreter 124 executes the verified bytecode 104 if it is not compiled by the JIT compiler 126.

The JIT compiler 126 compiles the verified bytecode 104 and generates native code.

The native code execution block 128 executes native code that is generated as a result of the compiled code process.

An explanation will now be given for one embodiment wherein the present invention is applied for the Java interpreter 124 and the JIT compiler 126.

The conditions governing the employment of this embodiment are as follows.

The Java interpreter 124 performs a transfer only upon receipt of a command that instructs a branch toward the end of a loop (a loop to be optimized, including a multiplex loop). A flag is set for a command that determines there will be no transfer, and a transfer is prohibited at a succeeding Thread.

It should be noted that after a transfer has been completed, program execution is resumed when a command is issued at the destination for the rearward branch performed in accordance with the command that initiated the transfer.

- 5 The Java interpreter 124 does not handle a transfer point at the transfer destination address, but upon the receipt of the rearward branching command that initiated the transfer.

10 In addition to the rearward branching command, issued when the Java interpreter 124 that activated the JIT compiler 126 determined that the transfer should be effected, the JIT compiler 126 detects, as a transfer point, another source for a rearward branching command for which a flag has not been set and that has not been optimized.

15 A loop surrounding a transfer point is not to be optimized, and thus the quantity of code that must be copied in order to move the transfer point outside the loop is reduced. This limitation is appropriate for the following reasons. If the surrounding loop is a "for" loop, the transfer is not currently performed because the Java interpreter 124 must determine the condition of the loop by permitting it to perform at least one iteration. Thus, the pertinent loop is assumed to be a loop having few iterations.

20 The transfer information is then generated as execution code.

To simplify the explanation, it should be noted that, in the code generated by the JIT compiler 126, the local variable area included in the original program is always allocated for a stack frame.

25 The following terms will be employed in this embodiment.

Transfer point: a point whereat program execution is transferred from the Java interpreter 124 to the JIT compiler 126.

30

Transfer bwd jump: a rearward branching command used by the Java interpreter 124 to detect a transfer point.

Transfer bwd jump address: an address in a memory wherein a bytecode transfer bwd jump command for a currently executing method is stored.

Transfer point table: a table wherein a transfer bwd jump address and the entry address for pad code are entered as a pair.

Method information (hereinafter simply referred to as mb): a Java structure in which method information is included.

Method compile information (hereinafter simply referred to as minfo): a structure for storing, during a compiled code process, various information concerning a method.

Process performed by the Java interpreter 124 (process 1)

The Java interpreter 124 detects a backedge by which it is immediately transferred to the JIT compiler 126.

If necessary, a value cached in the register is written in the memory.

The JIT compiler 126 is called to compile source code. At this time, the address of the backedge, which triggered the transfer, is transmitted to the JIT compiler 126.

Process performed by the JIT compiler 126 (process 2)

Locking is performed to provide an exclusion process for the compiling.

If the compiling has already been completed when the transfer bwd jump address is received, the transfer point table is examined. When the pertinent transfer point address has been included in the table, the table offset is transmitted as a return value (end).

- 5 When the transfer bwd jump address is received, in the method information a flag (hereinafter referred to as a gen\_tp flag) is set to represent the execution of a process for the transfer point.

The rearward branches included in the loop are counted, and the transfer point table is allocated.

10

The bytecode is traversed to build a basic block (hereinafter referred to as a BB) and a loop.

When the above flag is set, the following processing is performed.

15

The BB of the transfer point, which is obtained from the transfer bwd jump data received from the Java interpreter 124, is written in the method information.

A flag that represents the transfer point is set in the BB. The BB wherein this flag is set should not merge with the preceding BB.

20

At a rearward branch for forming the loop, a command for which a flag indicating no transfer is not set is entered in the transfer point table.

When a transfer point is included in the loop, the following processing is performed.

25

In accordance with the control flow graph, the code from the top of the loop to the transfer point is traced. During the tracing, whether the next command is present is determined. When the command is not present, the transfer point is moved to the top of the loop.

30

That is, in accordance with the control flow graph, the code from the top of the loop to the transfer point is traced.



Then, during the tracing process, whether the next command is present is determined. When the command is not present, the transfer point is moved to the top of the loop, and a command for writing data to the heap area is executed to end the processing for moving outside the loop.

5

Following this, a command for defining the local variable is executed.

Thereafter, the code that extends from the top of the loop to the transfer point is copied to a location outside the loop, and the transfer point is also moved outside.

10

The common sub-expression elimination process is performed.

When the process is extended across the transfer point, and when the calculation to be removed can be performed by a recalculation using a local variable belonging to the Java interpreter 124, the local variable is registered as recalculation information at the transfer point. When the recalculation can not be performed, the extension of the process across the transfer point is prohibited.

15

The privatization is performed for the field variable.

20

When the process is extended across the transfer point, and when the calculation to be removed can be performed by a recalculation using the local variable belonging to the Java interpreter 124, the local variable is registered as recalculation information at the transfer point. When the recalculation can not be performed, the extension of the process across the transfer point is prohibited.

25

The following processing is performed for code generation.

By using the information contained in the transfer point table for "minfo," a transfer point table wherein code (original code address, transfer code) is entered as an element and a table sized

30

storage area is prepared in the location that can be accessed by the mb. The bytecode address of the transfer point is then written.

The bytecode addresses are rearranged in the ascending order to facilitate the search.

The information contained in the transfer point table and the register image of the transfer point are employed to generate recalculation code for caching and a transfer code for each transfer point (code for storing a value in memory in a register). The generated codes are written in the transfer point table entered as transfer point code addresses.

A work area (a local variable area and a stack area) required by the JIT compiler 126 is established in a location that can be accessed by the mb.

Process performed by the Java interpreter 124 (process 3)

The return value for the compiler is examined. When the compilation failed, the execution of the Java interpreter 124 is continued (End).

The transfer point table for the mb is obtained to determine whether the table contains a current bwd jump address. When there is no address in the table, the execution of the Java interpreter 124 is continued. (End).

While taking into account the size of a work area received by the JIT compiler 126, the stack frame used for the Java interpreter 124 is converted into one for the JIT compiler 126.

The transfer point table is obtained from the mb.

The transfer point table is obtained, and the transfer point code address is also acquired from the address for the bytecode that is currently being executed.

The program execution is started at the transfer point code address.

The processing performed by the program execution apparatus 1 of the invention will now be described while referring to the flowcharts in Figs. 2 to 8.

5

The present invention is positioned as the process at S14 in the bytecode execution process (S10 in Fig. 2).

10

As is shown in Fig. 2, at step 100 (S100), the program execution apparatus 1 executes a method.

15

At step 102 (S102), the program execution apparatus 1 determines whether the method should be executed by the interpreter. When the method is to be executed by the interpreter, program control advances to S104. In the other case, program control is shifted to S114.

20

At step 104 (S104), the program execution apparatus 1 begins the processing using the interpreter.

At step 106 (S106), the program execution apparatus 1 reads a command.

At step 108 (S108), the program execution apparatus 1 determines whether compiled code is to be executed. When the code is to be executed, program control advances to S16 in S14. In the other case, program control is shifted to S110.

25

At step 110 (S110), the program execution apparatus 1 uses the interpreter to execute the code.

30

At step 112 (S112), the program execution apparatus 1 terminates the process that was using the interpreter.

At step 114 (S114), the program execution apparatus 1 uses the compiler to compile the method.

5 At step 116 (S116), the program execution apparatus 1 executes the native code that the compiler generated.

Fig. 3 is a flowchart showing the process at S14 according to the present invention.

10 As is shown in Fig. 3, at step 160 (S160) during the preprocessing (S16) for a transfer that is included in S14, the program execution apparatus 1 determines whether the method has been compiled. If the method has been compiled, program control advances to S162. In the other case, program control is shifted to S20, which will be described later while referring to Figs. 4 to 8.

15 At step 162 (S162), the program execution apparatus 1 obtains the transfer information table from the method.

20 At step 164 (S164), the program execution apparatus 1 examines the transfer data table, while using as a key the command address for the transfer point.

At step 142 (S142), the program execution apparatus 1 determines whether the transfer information is present. If the transfer information is present, program control advances to the transfer process (S18). In the other case, program control is shifted to S146.

25 At step 144 (S144), the program execution apparatus 1 determines whether the compiling performed at S20 was successful. If the compiling was performed successfully, program control advances to S18. In the other case, program control is shifted to S146.

30 At step 146 (S146), the program execution apparatus 1 returns to the execution process for which the interpreter is used.

At step 148 (S148), the program execution apparatus 1 terminates the processing.


At step 180 (S180) of the transfer process (S18), the program execution apparatus 1 obtains  
5 the transfer information from the transfer information table.

At step 182 (S182), the program execution apparatus 1 restores necessary information in a  
current stack frame.

10 At step 184 (S184), the program execution apparatus 1 changes the size of the stack frame by  
using the transfer information.

At step 186 (S186), the program execution apparatus 1 generates a stack frame for native code  
by using the transfer information and the restoring information.

15 At step 188 (S188), the program execution apparatus 1 obtains a transfer address by using the  
transfer information, and begins execution of the program. After the execution of the program  
has been completed, the program execution apparatus 1 terminates the processing (S148).

20  Fig. 4 is a flowchart showing the compiled ~~code~~ process (S20) for a transfer shown in Fig. 3.

At step 202 (S202) in S20, the program execution apparatus 1 divides the code of the method  
into basic blocks.

25 At step 204 (S204), the program execution apparatus 1 determines whether a transfer point is  
the head of the basic block. If the transfer point is the head, program control advances to S208.  
In the other case, program control is shifted to S206.

30 At step 206 (S206), the program execution apparatus 1 divides the basic block so that the  
transfer point is located at its head.

At step 208 (S208), the program execution apparatus 1 generates a control flow graph.

At step 210 (S210), the program execution apparatus 1 sets a mark in the basic block headed  
5 by the transfer point, and advances to the processing for searching for another proposed  
transfer point (S22 in Fig. 5).

Fig. 5 is a flowchart showing the processing (S22) performed in Fig. 4 when another proposed  
transfer point is searched for.

As is shown in Fig. 5, at step 222 (S222) in S22, the program execution apparatus 1 initiates an  
iteration process for code other than the transfer point in the method.

At step 224 (S224), the program execution apparatus 1 determines whether the process may be  
15 transferred from the interpreter. If such an action is not possible, program control advances to  
S226. In the other case, program control is shifted to S236.

At step 226 (S226), the controlling program analyzes the need for a transfer.

At step 228 (S228), the program execution apparatus 1 determines whether noticeable transfer  
20 effects can be provided. If noticeable effects can be provided by the transfer, program control  
advances to S230. In the other case, program control is shifted to S232.

At step 230 (S230), the program execution apparatus 1 determines whether the transfer point is  
25 located at the head of the basic block. If the transfer point is located at the head, program  
control advances to S234. In the other case, program control is shifted to S232.

At step 232 (S232), the program execution apparatus 1 divides the basic block so that the  
transfer point is located at the head of the basic block, and corrects the control flow graph.

At step 234 (S234), the program execution apparatus 1 sets a mark in the basic block of the transfer point.

At step 236 (S236), the program execution apparatus 1 terminates the iteration process for code other than the transfer point in the method.

At step 238 (S238), the program execution apparatus 1 terminates the processing at S22, and performs the processing (S26 in Fig. 6) for moving the transfer point outside the loop, without modifying the loop.

Fig. 6 is a flowchart showing the processing (S26) used to move the transfer point outside the loop, without modifying the loop in Fig. 4.

As is shown in Fig. 6, at step 262 (S262) in S26, for each transfer point the program execution apparatus 1 repeats up to S300 the processing for loop 1.

At step 264 (S264), the program execution apparatus 1 determines whether the transfer point is located in a loop that can be optimized. If the transfer point is located in such a loop, program control advances to S266. In the other case, program control is shifted to S300.

At step 266 (S266), the program execution apparatus 1 determines whether there is a writing command in the loop for an area that can be referred to by a component other than the method. If such a writing command is present in the loop, program control is shifted to S300. In the other case, program control advances to S268.

At step 268 (S268), the program execution apparatus 1 calculates a set  $V_r$  of local variables whose values, which are defined outside the loop, are referred to in the loop.

At step 270 (S270), the program execution apparatus 1 calculates a set  $V_d$  of local variables that are defined in the loop.

SUB A7 At step 272 (S272), the program execution apparatus 1 repeats the processing for loop 2 up to S282 for each variable VI in a set (Vr 3 Vd) of local variables that require recalculation.

5 At step 274 (S274), the program execution apparatus 1 collects information required for a recalculation of the variable VI at the loop start point.

At step 276 (S276), the program execution apparatus 1 determines whether recalculation of the variable VI can be performed at the loop start point. If recalculation of the variable VI can be performed, program control advances to S278. In the other case, program control is shifted to S300.

10 At step 278 (S278), the program execution apparatus 1 calculates a set Vc of local variables that are required for a recalculation of the variable VI.

15 SUB A8 At step 280 (S280), the program execution apparatus 1 determines whether the set (Vc 3 Vd) is empty. If the set (Vc 3 Vd) is empty, program control advances to S282. In the other case, program control is shifted to S300.

20 At step 284 (S284), the program execution apparatus 1 moves the transfer point to the loop start point.

SUB A9 At step 286 (S286), the program execution apparatus 1 determines whether the set (Vr 3 Vd) is empty. If the set (Vr 3 Vd) is empty, program control advances to S300. In the other case, program control is shifted to S288.

At step 288 (S288), the program execution apparatus 1 generates one basic block.

At step 290 (S290), the program execution apparatus 1 generates a route for a control flow that extends from the newly generated basic block to the loop start point.




At step 292 (S292), the program execution apparatus 1 changes the route leading from outside the loop to the loop start point into a route that leads to the newly generated basic block.

5 At step 294 (S294), the program execution apparatus 1 repeats for recalculation of each local variable the processing extending from loop 3 up to S298.

At step 296 (S296), the program execution apparatus 1 generates recalculation code using the recalculation information, and adds the obtained code to the code in the newly generated basic block.  
10

At step 302 (S302), by modifying the loop, the program execution apparatus 1 advances to the processing (S32 in Fig. 7) for moving the transfer point to a position outside the loop.

15  Fig. 7 is a flowchart showing the processing (S32) for using the loop modification shown in Fig. 4 to move the transfer point outside the loop.

As is shown in Fig. 7, at step 322 (S322) in S32, for each transfer point the program execution apparatus 1 repeats the processing for loop 1 up to S234.  
20

At step 324 (S324), the program execution apparatus 1 determines whether the transfer point is located in a loop that can be optimized. If the transfer point is located in such a loop, program control advances to S286. In the other case, program control is shifted to S334.

25 At step 326 (S326), the program execution apparatus 1 obtains a basic block BBp in a loop that immediately post-dominates the loop start point and the transfer point.

At step 328 (S328), the program execution apparatus 1 copies a control flow graph that is formed using the sum of the routes from the loop start point to the basic block BBp.  
30

At step 330 (S330), the program execution apparatus 1 changes all the control flows that extend from outside the loop to the loop start point into control flows in a flow graph copy that extends to the basic block that corresponds to the loop start point.

5 At step 332 (S332), the program execution apparatus 1 maintains the control flows extending from the original control flow graph to the basic block BBp.

At step 336 (S336), the program execution apparatus 1 returns to S208 in S20 in Fig. 4.

10 An explanation for the processing will be given by referring again to Fig. 4.

At step 208 (S208), while taking a transfer point into account, the program execution apparatus 1 performs redundancy elimination.

15 At step 210 (S210), the program execution apparatus 1 performs optimization for which a transfer point need not be taken into account.

20 ~~At step 212 (S212), the program execution apparatus 1 generates native code for the method, and advances to the processing (S34 in Fig. 8) for generating transfer information.~~

~~Fig. 8 is a flowchart showing the processing (S34) for generating transfer information in Fig. 4.~~

At step 342 (S342) in S34, the program execution apparatus 1 stores a stack frame, having the size that is required by the compiled code, at a location that can be accessed by the method.

25 ~~At step 344 (S344), the program execution apparatus 1 repeats for each transfer point the processing performed up to S328 for loop 1.~~

30 At step 346 (S346), the program execution apparatus 1 generates transfer information (the code address of a transfer point, cache recalculation information obtained by redundancy

elimination, register use state information for a transfer point, and the address of a command by which the interpreter determines whether a transfer is to be made).

At step 350 (S350), the program execution apparatus 1 returns to S144 in Fig. 3.

#### Effects provided by the program execution apparatus 1

A "method whereby the performance is greatly affected by transferring the program execution in the middle of the processing" that is optimized by the program execution apparatus 1 is, for example, a method that includes an endless loop, or a method wherein multiple iterations of a loop are performed that produce a difference in the priorities existing between the execution of the interpreter and the execution of the compiled code.

The code must be scanned before the execution in order to examine, without program execution, whether a loop is included. However, when there is a great amount of code, the time required for the scanning may cause the performance to be deteriorated.

While executing the program, the program execution apparatus 1 can detect a loop without scanning being required, and can transfer the interpreter to the compiled code to enable fast processing.

Generally, at an arbitrary point the interpreter is transferred to the compiled code, and in situations where transfers should be performed, in most cases transfer points are present in loops. However, since it is difficult to optimize a loop in which there is a transfer point, the processing for moving the transfer point outside the loop is an important performance enhancing procedure.

#### Advantages of the Invention

